社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－FI－77 (6)
2004／11／26

透過的アクセスが可能なXML文書圧縮法

中西 優介† 白井 靖人‡

近年、柔軟なデータ表現力と拡張性を持ったXMLの利用が広まってきている。それにともないXML文書を扱うアプリケーションも増加してきている。一般的にXML文書は可読性を保つために複数の文字からなる意味のある要素名を用いる。そのためにXML文書自体のサイズが大きくなってしまう。したがって、XML文書の圧縮が必要になる。
しかし、従来の圧縮方法ではXML文書をバイナリーコードにするために圧縮状態のXML文書を操作することができない。本稿では、圧縮状態のXML文書に対して透過的なアクセスを可能にする方法について提案する。それによって、圧縮状態のXML文書に対してもDOMやXSLT等による操作が可能になる。

The XML document compressing method which can be transparent access

Yuusuke Nakanishi† Yasuto Shirai‡

In recent years, the use with flexible data power of expression and flexible extendibility of XML has spread. The application which treats an XML document according to it has also been increasing. Generally, an XML document uses an element name with the meaning consist of two or more characters, in order to maintain readability. Consequently, the size of the XML document itself will become large. Therefore, compression of an XML document is needed.
However, in the conventional compression method, in order to make an XML document into a binary code, the XML document of a compression state cannot be operated. In this paper, it proposes about the method of enabling transparent access to the XML document of a compression state.By it, operation by DOM, XSLT, etc. can be performed also to the XML document of a compression state.

1. はじめに

近年、柔軟なデータ表現力と拡張性を持ったXML[1]の利用が広まってきている。それにともないXML文書を扱うアプリケーションも増加してきている。

一般的にXML文書は可読性を保つために、複数の文字からなる意味のある名前を要素名や属性名として用いる。そのために扱う情報量が増加するにしたがってXML文書中における要素名および属性名の占める割合も大きくなり、XML文書自体のサイズが大きくなる。その結果、巨大なXML文書に対して圧縮が必要となる。

現在、XMLの圧縮技術にはxmill[3]やXMLZIP等が提案されている。

しかし、これら提案されている圧縮方法は高い圧縮率を得るためにXMLの規格に基づいていない独自のバイナリ形式にXML文書の変換を行っている。その結果、圧縮状態のXML文書に対してDOM[4]による操作や問い合わせなどXMLの処理系を利用することが不可能となる。

そこで、本稿ではXMLの規格を損なわない圧縮を行うことで圧縮状態のXML文書に対してもDOMによる操作や問い合わせなどを可能にする圧縮方法の提案を行う。そうすることにより、圧縮されたXML文書をXMLの処理系によって利用する際に圧縮されているということを意識せずに利用が可能になる、つまり透過的アクセスも可能になる。

2. 関連技術

XML文書の圧縮技術にはxmillやXMLZIPが提案されている。これらの圧縮技術ではgzip圧縮を用

†静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
‡静岡大学情報学部情報学科
Faculty of Informatics, Shizuoka University

い、要素ごとに最適な圧縮を行うことにより、XML 文書全体を gzip で圧縮する以上に高い圧縮率を得ている。具体的に圧縮率で表すと、元の XML 文書の 10 分の 1 のサイズに圧縮する。

しかし、この圧縮方法により圧縮された XML 文書は XML の規格にしたがっていない独自のバイナリ形式に変換されている。したがって、xmill により圧縮された XML 文書に対して DOM による操作や問い合わせをすることができない。また、XMLZIP により圧縮された XML 文書は専用の DOM 関数を用意することで DOM による操作は可能になっているが問い合わせができない。

3. 圧縮・伸張方法

3.1 圧縮

xmill や XMLZIP といった従来の圧縮方法で、圧縮状態の XML 文書を扱うことができないは、XML 文書を XML の規格にしたがっていない独自の形式に変換してしまうためである。

そこで本稿では、XML 文書中に現れる要素名及び属性名を短い文字列に置換を行うことで XML の規格を損なわない XML 文書の圧縮を行う。

圧縮は以下の 5 つの手順で行う。ここで、圧縮名とは、圧縮属性名と圧縮属性名の総称である。

手順 1 スキーマの解析
手順 2 要素名・属性名の出現頻度の算出
手順 3 圧縮名の生成
手順 4 要素名及び属性名の置換
手順 5 対応表の作成

手順 1 により、XML 文書の構造が記述されているスキーマの解析を行う。本稿では、DTD と XML Schema[2]の解析を行えるようにする。DTD 及び XML Schema による要素宣言と属性宣言は Fig.1 に示す形式で行われている。

各宣言の形式に従って、XML 文書中に現れるすべての要素名及び属性名とそれぞれの文字数を解析する。

手順 2 により、XML 文書中に現れる各要素及び属性の出現頻度を求める。

出現頻度のカウント方法は、開始タグと終了タグの両方で 1 回とカウントするのではなく別々でカウントする。例えば、「<name>Tom</name>」という要素は、「name が 1 回現れた」とカウントするのではなく、「name が 2 回現れた」とカウントする。

出現頻度を求めた後には、要素の文字数とその出現回数の積、属性の文字数とその出現回数の積をそれぞれ求め、要素と属性それぞれその値の大きい順に順位付けを行う。

手順 3 により、手順 2 により与えられた順位付けの順に圧縮名の生成を行う。圧縮要素名の生成の手順は以下の通りである。

- XML 文書中に現れる要素が 53 種類以下の場合は、手順 2 において順位付けされた順位の高い順にアルファベットの順で a から z、A から Z、下線(_)を圧縮要素名として割り当てる。
- XML 文書中に現れる要素が 53 種類を超える場合、54 番目の要素以降 aa、ab といったように XML の規格に基づいておりなおかつできる限り短い文字列を圧縮要素名として割り当てる。
- XML の規格上、コロン(:)も圧縮要素名の一部として使用することができるが、“Namespaces in XML”において特別な意味が与えられているので使用しない。

属性についても、圧縮要素名を生成した手順と同様の手順で圧縮属性名の生成を行う。

Fig.2 の XML 文書に対してこれまでの手順を適用すると Fig.3 のような圧縮名を生成することができる。

DTD による要素宣言

<!ELEMENT 要素名 制約条件 >

XML Schema による要素宣言

<xsd:element ... name="要素名" ... >

DTD による属性宣言

<!ATTLIST 要素名 属性名 属性の型 デフォルト宣言.
属性名 属性の型 デフォルト宣言
>

XML Schema による属性宣言

<xsd:attribute ... name="属性名" ... >

Fig.1 DTD 及び XML Schema よる各宣言の形式

手順4により、XML文書中の要素名及び属性名を手順3により生成した圧縮名に置換する。

また、圧縮されたXML文書中のroot要素の前に手順5で作成する対応関係を表わしたファイルのある場所をコメントとして挿入する。

Fig.2のXML文書を本研究で提案した圧縮方法で圧縮を行うとFig.4のようになる。

手順5により、置換前の要素名及び属性名と圧縮名との対応関係がわからないと圧縮されたXML文書を伸張することができないので、その対応関係を表わしたファイルを作成する。

要素名及び属性名と圧縮名との対応関係は、Fig.5で表わしたXMLの形式で行う。Fig.5の詳細は次の通りである。

- 要素 elem には圧縮要素名と置換前の要素名との対応関係を記述する
- 要素 attr には圧縮属性名と置換前の属性名との対応関係を記述する
- 属性 a_name には圧縮要素名または圧縮属性名を記述する
- 属性 b_name には置換前の要素名または置換前の属性名を記述する

```
<accounts type='1'>
  <co-owner id='1'>John Doe</co-owner>
  <co-owner id='2'>Jack Smith</co-owner>
  <checking>
    <balance>170.00</balance>
    <transaction>-100.00</transaction>
    <transaction>-500.00</transaction>
    <fee>4.00</fee>
  </checking>
  <co-owner id='1'>John Doe</co-owner>
  <savings>
    <balance>5000.00</balance>
    <interest>212.50</interest>
  </savings>
</accounts>
```

Fig.2 XML文書のサンプル

要素名	文字数	回数	積	圧縮名
co-owner	8	6	48	a
transaction	11	4	44	b
balance	7	4	28	c
accounts	8	2	16	d
checking	8	2	16	e
interest	8	2	16	f
savings	7	2	14	g
fee	3	2	6	h

属性名	文字数	回数	積	圧縮名
id	2	3	6	a
type	4	1	4	b

Fig.3 圧縮名の生成

```
<!--http://sample/xml/table.xml-->
<d b='1'>
  <a a='1'>John Doe</a>
  <a a='2'>Jack Smith</a>
  <e>
    <c>170.00</c>
    <b>-100.00</b>
    <b>-500.00</b>
    <h>4.00</h>
  </e>
  <a a='1'>John Doe</a>
  <g>
    <c>5000.00</c>
    <f>212.50</f>
  </g>
</d>
```

Fig.4 Fig2 の XML 文書を圧縮した XML 文書

要素名と圧縮要素名との対応関係

<elem a_name="…" b_name="…">

属性名と圧縮属性名との対応関係

<attr a_name="…" b_name="…">

Fig.5 記述形式

```
<root>
<elem  a_name="a" b_name=" co-owner"/>
<elem  a_name="b" b_name=" transaction"/>
<elem  a_name="c" b_name=" balance"/>
<elem  a_name="d" b_name=" accounts"/>
<elem  a_name="e" b_name=" checking"/>
<elem  a_name="f" b_name=" interest"/>
<elem  a_name="g" b_name=" savings"/>
<elem  a_name="h" b_name=" fee"/>
<attr  a_name="a" b_name=" id"/>
<attr  a_name="b" b_name=" type"/>
</root>
```

Fig.6 対応関係表

Fig.4 の圧縮された XML 文書の要素名と圧縮要素名の対応関係のファイルは Fig.6 のようになる。

3.2 伸張

伸張の手順は、圧縮よりも単純で次の二つの手順から成る。

手順 1　対応表の解析
手順 2　圧縮名の置換

手順 1 により、圧縮名に対応する属性名及び要素名を解析する。解析は対応表作成のときに記述した記述規則にしたがって行う。

手順 2 により、圧縮された XML 文書中の圧縮要素名及び圧縮属性名を手順 1 の対応表の解析によって得られた結果にしたがって要素名及び属性名に置換を行う。

また、置換後には圧縮の際に XML 文書中に挿入したコメント(対応関係を表わしたファイルのある場所)を削除する。

4. 透過的アクセス

提案した圧縮方法により圧縮状態のまま XML 文書にアクセスすることは可能になったが、透過的なアクセスを行うことはできない。このままではユーザが圧縮された XML 文書に対して操作や問い合わせを行うためには、圧縮名と元の要素名、属性名との対応関係を意識しなければならない。

そこで透過的アクセスを可能にするためには XML 文書の処理系を若干改良する必要がある。具体的には、操作を行う時や問い合わせの結果を出力する際に、置換前の要素名や属性名を圧縮名に置換，またはその逆の処理を自動的に行うプログラムを追加する必要がある。

本稿では、プログラムの実装方法の例としていくつかの処理系を例にあげて説明する。

4.1 DOM

DOM は、XML の操作を行うための標準インタフェースである。DOM では、パーサによって XML 文書を木構造に表す。木の各ノードには要素・属性・データなどの情報が格納されており、要素名や属性名が各ノードのノード名となっている。

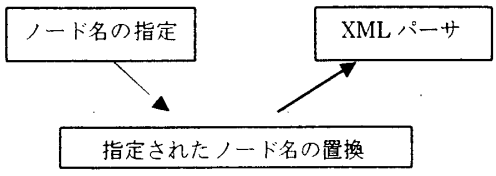
したがって、圧縮状態の XML 文書の操作を行う時に置換前の要素名や属性名を使ってノード名を指定してもノードの操作を行うことができない。また、ノード情報の取得を行っても圧縮状態の XML 文書の情報がえられるだけなので DOM の利用者が必要とする情報ではないので意味がない。よって、DOM による透過的アクセスを可能にする必要がある。

本稿では、Apache XML Project の一部として開発されている XML パーサとして Xerces☆を透過的アクセスが可能になるようにする。

しかし、Xerces ではパーサの部分のソースコードは配布されていないので完全な透過的アクセスはできない。つまり、DOM インタフェースを利用する際に置換前の要素名及び属性名と圧縮名との対応関係を表わしたファイルのある場所の指定をユーザが行う必要がある。

4.1.1 ノードの操作

ノードの操作を透過的に行うためには、DOM の利用者によって指定されたノード名を XML 文書の圧縮のときに作成した対応表と照らし合わせて自動的に置換することで透過的なノードの操作が可能になる。



ノード操作においてプログラムの追加の必要があ

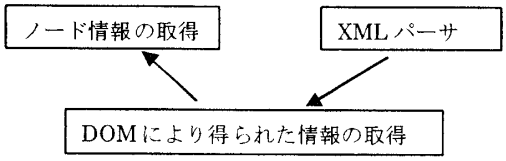
☆http://xml.apache.org/xerces2-j/

るインタフェースは次のものである。

- Document
- Element
- NamedNodeMap

4.1.2 ノード情報の取得

ノード情報を透過的に取得するためには、DOM パーサから得られた情報を DOM の利用者に渡す前に、DOM パーサから得られた情報を対応表と照らし合わせて置換してやることで正しいノード情報を得ることが可能となる。



ノード情報の取得においてプログラムの追加の必要があるインタフェースは次のものである。

- Attr
- Element
- NamedNodeMap

4.2 XSLT

DOM よりも高レベルなアクセスを行う例としてまず XSLT[5]を取り上げる。

XSLT は、XML によって記述された文書をほかの XML 文書に変換するための言語である。XSLT では、XML 文書中の位置指定の記述方式として Xpath が使われる。

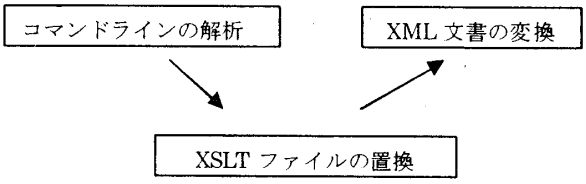
例えば、Fig.2 のような XML 文書で「balance」という要素名を Xpath で指定するためには「accounts/checking/balance」と記述する。つまり、Xpath による位置の指定には置換の対象となる XML 文書中の要素名や属性名を用いられる。

したがって、圧縮前の XML 文書中の要素名や属性名を用いた XSLT ファイルでは圧縮状態の XML 文書の変形を行うことができないので透過的アクセスを可能にする必要がある。

本研究では、Apache XML Project の一部として開発されている XSLT プロセッサの Xalan☆において、透過的アクセスができるようにする。

Xalan では一般的にコマンドラインを用いて XML 文書の変換を行う。XSLT プロセッサは、コマンドラインで与えられた変換対象の XML 文書と XSL ファイルを解析し、その後に XSL ファイル内に記述されている規則にしたがって XML 文書の変換を行う。

よって、Xalan で XSLT を利用する際に透過的アクセスを可能にするためには、XSLT プロセッサ内でコマンドラインの解析を行った後 XML 文書の変換処理に移る前に、XSL ファイル中の Xpath で記述された部分を対応表に照らし合わせて自動的に置換を行えば透過的なアクセスが可能となる。



具体的な手順は、以下の通りである。

手順 1 対応関係を表わしたファイルの場所の特定
手順 2 対応関係の解析
手順 3 XSLT ファイル内の要素名及び属性の置換

手順 1 により、対応関係を表わしたファイルの場所は圧縮された XML 文書中のコメントより特定する。

手順 2 により、置換前の要素名及び属性名と圧縮名との関係を解析する。

手順 3 により、XSLT ファイル中で Xpath を用いて記述されている要素名及び属性名を圧縮名に置換する。

また、XSLT での処理が行われた後には、XML 文書を変形するために要素名及び属性名の置換が行われた XSLT ファイルを元の状態に戻しておく必要がある。

4.3 XQuery

XSLT のほかに DOM よりも高いレベルで XML 文書へアクセスを必要とする例として、XQuery[6]があげられる。

XQuery は、問い合わせ言語の一つであり、XML 文書中の位置の指定を行う際は、XSLT と同様に

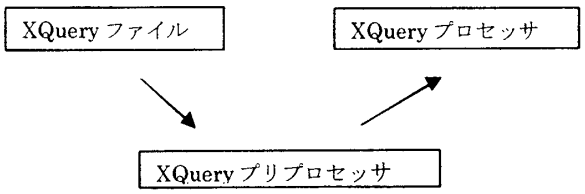
☆ http://xml.apache.org./xalan-j/

Xpathを用いて行われる。

したがって、圧縮前のXML文書中の要素名や属性名を用いたXpathで記述されているXQueryファイルでは圧縮状態のXML文書に対して問い合わせを行うことができないので透過的アクセスを可能にする必要がある。

本稿ではまだ透過的アクセスを可能にするためにどのXQueryプロセッサにどのように実装するか決めていない。しかし、透過的アクセスを可能するためには次のようにプログラムの実装をすればよいと考えている。

実装方法は、XQueryファイルをXQueryプリプロセッサに渡し、XQueryファイル中で置換の必要がある部分を対応表に照らし合わせて置換を行ってからXQueryプロセッサにXQueryファイルを渡すことで透過的な問い合わせが可能になると考えている。また、問い合わせた結果に対しても必要な部分を置換することでXQueryによる透過的アクセスが可能になると考えている。



XQueryプロセッサによるXML文書への問い合わせが行われた後には、XSLTの場合と同じくXQueryファイル中で置換が行われた要素名及び属性名を元の要素名及び属性名に戻しておく必要がある。

5. まとめと今後の課題

本稿では、XML文書が圧縮されている状態でもXMLの規格を損なわない圧縮方法を提案することで圧縮状態のXML文書に対してもXMLの処理系が利用可能になった。また、XMLの処理系に若干の改良を加えることで圧縮しているということを意識せずにXML文書の操作や問い合わせができる、つまり透過的アクセスが可能になった。

また今後の課題は、まだ実装できていないXQueryによる透過的アクセスを可能にすること本稿であげた処理系以外で透過的アクセスが必要となる処理系の考察があげられる。

参考文献

[1]W3C,"Extensible Markup Language (XML) 1.0 (Second Edition)"
http://www.w3.org/TR/REC-xml
[2]XML Schema : http://www.w3.org/XML/Schema
[3]H.Liefle,et al.:Xmill:an Efficient Compressor for XML Data,Sigmod Record Vol.29 No.2.June 2000
[4] Document Object Model(DOM) http://www.w3.org/DOM
[5] XSL Transformations (XSLT) : http://www.w3.org/TR/xslt
[6] XQuery 1.0: An XML Query Language http://www.w3.org/TR/xquery/